UNIVERSIDADE FEDERAL DO PARÁ

INSTITUTO DE TECNOLOGIA

FACULDADE DE COMPUTAÇÃO E TELECOMUNICAÇÕES

ENGENHARIA DA COMPUTAÇÃO

Projeto de Hardware e Interfaceamento: UART utilizando o FPGA

Adriano Madureira dos Santos – 201706840041

João Vitor Daltro Tavares Paiva -201606840076

Victor Hugo Gonçalves Pantoja - 201706840003

Wendel Williams Cardoso dos Santos - 201606840039

BELÉM-PA

2019

Sumário

1. Introdução

Este trabalho tem por intuito prover o interfaceamento entre o computador e FPGA (*Field Programmable Gate Array*) através do protocolo de comunicação serial UART (Universal Asynchronous Receiver/Transmitter) em que serão enviados informações do PC em um dos *baud rates* especificados, neste caso, serão utilizados 2 *baud rates* distintos um 9600 e outro 115200 ambos medidos em bits/s selecionados por uma chave do FPGA. Além disso, serão utilizados códigos em VHDL (VHSIC Hardware Description Language) para poder sintetizar o transmissor (tx), receptor (rx) e um código para seleção de *baud rate.* Desta forma, as informações transmitidas pelo computador ao FPGA na comunicação UART devem ser apresentados nos LEDs vermelhos da placa FPGA.

2. Desenvolvimento

2.1. UART

UART (Universal Asynchronous Receiver/Transmitter) é um protocolo, ou tecnologia de hardware de um computador que permite a comunicação serial assíncrona em que o formato de dados e a velocidade são configuráveis, com isso não existe um controle por sinal de clock sendo enviado entre as duas entidades que estão se comunicando, ou seja, cada uma das entidades devem saber quando visualizar, fazer a leitura daquele bit, o sinal sendo um bit 0 ou 1. Além disso, como o próprio nome exemplifica é um protocolo universal, isto é, uma gama de dispositivos o utiliza. Assim, o UART é um CI usado para comunicações seriais em uma porta do computador ou dispositivo periférico em que sua sinalização elétrica é feita por dispositivos externos, isto é, as entidades que desejam fazer uma transmissão e recepção de dados. Neste aspecto, este protocolo não é usado diretamente na geração e recepção de sinais externos de equipamentos, existem outros dispositivos que fazem essa conversão desses sinais lógicos para os sinais externos dos equipamentos, um exemplo claro disso seria o Arduíno, placa de desenvolvimento, em que a comunicação dele é UART, entretanto é convertida para USB para ter a comunicação com outro dispositivo que é o computador.

Neste protocolo a comunicação é feita basicamente com dois pinos, um para transmitir dados e um para recebê-los sendo feita de forma *full-duplex* (bidirecional simultâneo), demonstrando a economia de pinos propiciada pela comunicação serial. O UART é usado para conversão de dados paralelos para seriais, os dados entram no UART se tornam seriais e em outro dispositivo UART no destino voltam a ser paralelos.

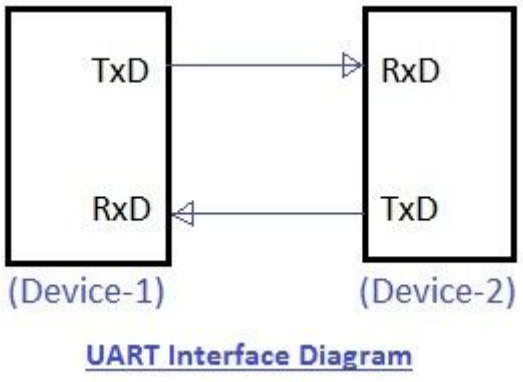

Figura 1 - Diagrama da interface do protocolo UART

O processo de uma comunicação UART é como mostrado na Figura 2 é descrita por iniciar em um estado inativo, ou *Bus Idle*, mantido em nível lógico alto, pois em questões históricas isso sinaliza que o equipamento está funcionando, toda vez que uma informação for enviada ela inicia com um *start bit*, em nível baixo, então seguem 8 bits de dados, incluindo os bits de paridade, mecanismo de ajuda na detecção de erros de envio, que termina com um *stop bi*t, em nível alto, esses bits são bits de sinalização de início e final da informação, não são dados úteis (informação de fato). Ademais, como o protocolo é assíncrono as leituras dos bits são feitas durante a duração de cada *baud* (tempo de espera de bit). Outro aspecto dessa comunicação digno de nota seria as entidades que utilizam o UART devem ter velocidades.

Figura 2 - Data Framing UART.

2.2. FPGA

O FPGA é um dispositivo lógico programável que permite implementar circuitos digitais em uma única placa de desenvolvimento, este circuito programável permite simular qualquer outro circuito fazendo dele um atrativo para simulação e desenvolvimento. Atualmente, os principais fabricantes são da Altera e Xilinx que promovem, devido ao desenvolvimento desses circuitos avanços em diversas áreas como: setor elétrico, telecomunicações, automotivo, entre outros. Neste projeto foram utilizados o kit da Altera com a placa da

família Cyclone II e sua programação foi feita pelo software Quartus 13.1 para compilação dos programas em VHDL que serão programados na placa.

2.3. Descrição detalhada do projeto.

Para a construção do projeto foram necessários duas descrições dentro do top-level, o RX e o TX, que serão explicados mais à frente. Para o UART, é preciso um sinal de clock de 50 MHz (PIN_L1), e chaves seletoras de reset (PIN_M1), baudrate (PIN_L2), oito chaves de entrada de dados pelo TX (PIN_L22, PIN_L21, PIN_M22, PIN_V12, PIN_W12, PIN_U12, PIN_U11, PIN_M2), botão de envio desses dados pelo TX (PIN_R22) e por fim uma entrada de dados seriais RX na porta (PIN_B13).

2.4. Diagrama de blocos

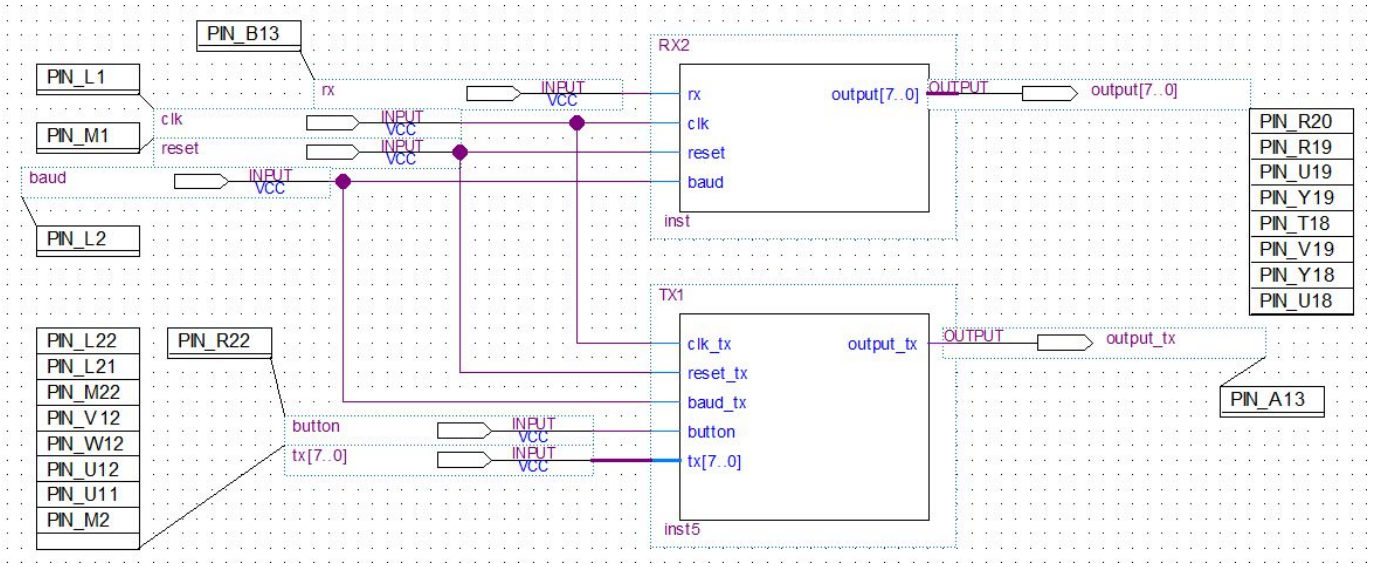

Figura 3 - Diagrama em Blocos.

Na arquitetura do UART, o *clock* será comum entre o TX (Transmissor de dados) e o RX (Receptor de dados), assim como o *reset* (reseta o programa para o estado inicial) e o *baud rate* (velocidade). O TX receberá em paralelo pela entrada “tx” e então enviará em modo serial pela saída “output_tx”, assim como o RX receberá dados, neste projeto foi utilizada a IDE do Arduino para envio de dados em caracteres, em série pela entrada “rx” e transformará para paralelo na saída “output”.

Essa arquitetura em blocos da Figura 3 é abstração do diagrama da Figura 1 sobre UART em que temos um transmissor e receptor em dois blocos distintos fazendo suas operações simultaneamente, blocos que serão o nível de abstração mais alto do projeto de comunicação UART, é válido salientar que cada bloco desse diagrama possui uma codificação em VHDL o qual será responsável por descrever e sintetizar os comandos necessários para fazer o interfaceamento entre o FPGA e computador.

2.5. Diagramas de Estados.

2.5.1. RX

Inicialmente o estado inicial é s0, é onde irá acontecer a detecção do bit start, o bit 0, assim quando rx, que é a entrada for igual 0 a máquina passa para o próximo estado. Em s1 iremos contar meio período de bit, na qual para a taxa de 9600 bps é 2600 ciclos e para 115200 é 216. Cálculo do período necessário:

50MHz / 9600 = 5200, 5200 /2 = 2600.

50MHz / 115200 = 432, 432/ 2 = 216.

Feito o meio periodo, é passado para o estado seguinte, s2, onde contamos mais um período inteiro, assim colhendo uma amostra no meio do bit de entrada RX, então segue-se ao próximo estado, s3, que por sua vez irá armazenar a entrada RX em um registrador de deslocamento e incrementando um contador de limite 8, pois pela definição os pacotes UART contém 8 bits de dados, quando nbits = 8 a máquina assume o estado s5 que contará mais um período inteiro para chegar a metade do bit start do pacote, ou seja, quando o contador chegar a 5200 ciclos ou 432 no caso de baud = 1, significando uma transferência a 115200 bps. O estado s4 assume e verifica se a entrada é igual a 1, o bit de parada, sendo verdadeiro, então volta-se ao s0 para começar tudo novamente.

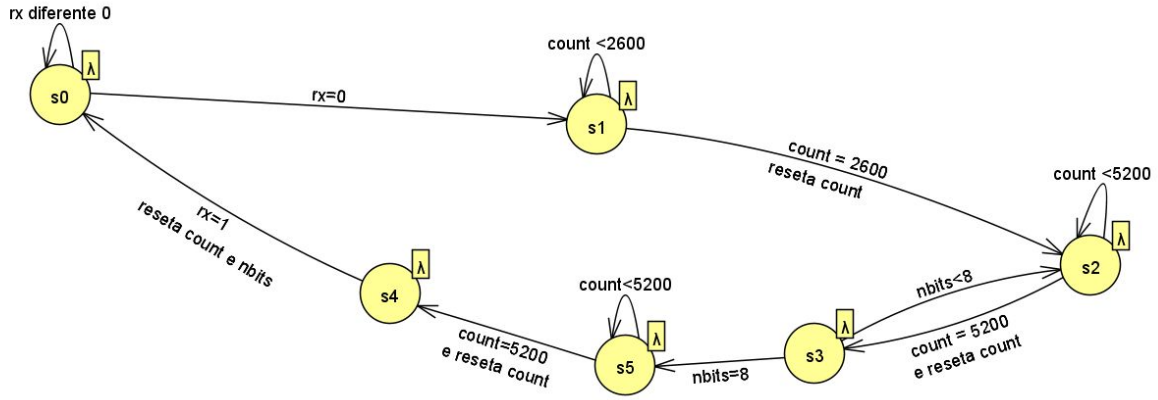

Figura 4 - Máquina de Estados de RX.

2.5.2. TX

Primeiramente o código se inicia no estado s5, onde basicamente ocorre o Bus Idle que está relacionado à idéia da espera pelo bit start, que neste caso é indicado pelo baixo nível lógico na linha serial de dados TX e ocorre quando o botão é pressionado. A passagem do estado s5 para o s6 é definida a partir do pressionamento do botão e uma consequente leitura de uma borda de subida. No estado s6 ocorre a transferência dos dados das entradas paralelas para o buffer do TX e em seguida ele coloca de fato o nível lógico baixo na variável output_tx indicando o bit start. Feito isso, o próximo estado, que é o s7, contará um período inteiro de bit e partir disso irá para s8, que usando um registrador de deslocamento, vai inserindo a entrada paralela na entrada serial oito vezes em um intervalo de um período de bit, após isso o estado sA contará mais um período para que no estado sB a saída seja colocada em nível lógico alto, indicando o bit stop para retornar ao estado inicial s5.

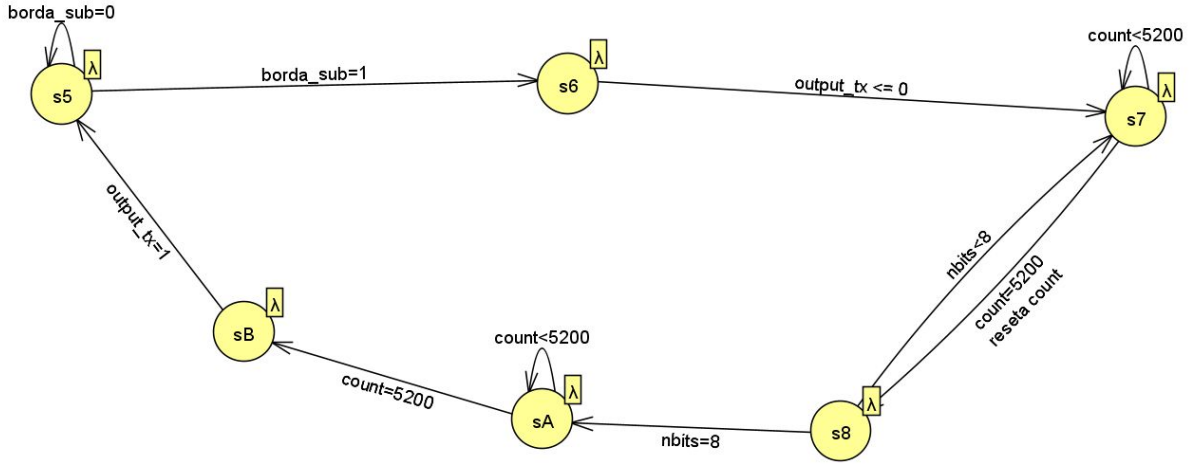

Figura 5 - Máquina de Estados de TX.

2.5.3. Considerações complementares

O código está estruturado em forma de máquina de estados de Moore, onde cada acontecimento está possivelmente mapeado. Dentro do *top level* encontra-se o código do *receiver*, RX, essa descrição se define como na Figura 6 , onde caso o baud seja 0 (if baud = 0), que pode ser definido a partir de uma chave seletora, então o código entrará na máquina de estados, definido na seção 2.5.1, com um divisor de frequência para um *baud rate* de 9600 bps, senão (else) entra-se em outra máquina de estados, semelhante a anterior, porém a taxa fica a 115200 bps alterando assim seu divisor de frequência, fora isso as máquinas de estados de TX e RX são as mesmas.

Para o TX, iremos construir o pacote para ser transmitido, como no RX, usamos uma máquina de estados como na Figura 6 para a seleção de taxa de *baud rate*. Primeiramente, o TX envia dados do FPGA para o computador conectado a ele semelhante a descrição feita na seção 2.5.2, vemos que para enviar informação é necessário o acionamento de um botão, sendo que a mensagem enviada é configurada nas chaves da placa. Ademais, o código funciona de acordo com a máquina de estados, e dessa forma, enviamos caracteres em ASCII, oito bits, para leitura no monitor serial da IDE Arduíno. Vale salientar que esse sinal de saída de TX é armazenado em um sinal chamado “buff” que será uma uma espécie de variável auxiliar para a manipulação do sinal. Os dados nesse bloco em VHDL são serializados a partir de um registrador de deslocamentos em que “nbits” são enviados a cada *baud*. No caso do RX as funções dessas variáveis como “buff” e “nbits” são semelhantes e seu funcionamento é baseado na máquina de estados descrita no tópico 2.5.1.

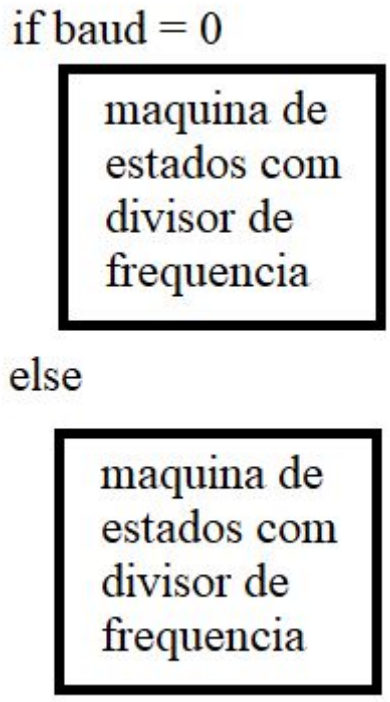

Figura 6 - Diagrama do Funcionamento do *Baud rate*.

2.6. Códigos

2.6.1. RX

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity RX2 is
port(rx,clk,reset,baud : in STD_LOGIC;
    output : out STD_LOGIC_VECTOR (7 downto 0));
end RX2;



architecture rtl of RX2 is

type state_type is (s0,s1,s2,s3,s4,s5);
signal state : state_type:= s0;
SIGNAL count: INTEGER RANGE 0 TO 5199 := 0;
SIGNAL nbits: INTEGER RANGE 0 TO 8 := 0;
SIGNAL buff : STD_LOGIC_VECTOR (7 downto 0):="00000000";
```

```
begin

output <= buff;

process(clk,reset)
begin
	if reset ='1' then
		state <=s0;
		buff<="00000000";
		count <=0;
		nbits<=0;
	elsif (rising_edge(clk)) then
			if baud = '0' then
				case state is
					when s0 =>
						if rx = '0' then
							state <= s1;
						else
							state <= s0;
						end if;
					when s1 =>
						if count /= 2599 then
							state <= s1;
							count <= count + 1;
						else
							state <= s2;
							count <= 0 ;
						end if;
					when s2 =>
						if count /= 5199 then
							state <= s2;
							count <= count + 1;
						else
							state <= s3;
							count <= 0 ;
						end if;
					when s3 =>
						if nbits < 8 then
```

```
                        state <= s2;
                        buff(6 downto 0)<=buff(7 downto 1);
                        buff(7)<=rx;
                        nbits<= nbits+1;
                    else
                        state <= s5;
                    end if;
                when s5 =>
                    if count /= 5199 then
                        state <= s5;
                        count <= count + 1;
                    else
                        state <= s4;
                        count <= 0 ;
                    end if;
                when s4 =>
                    if rx = '1' then
                        state <= s0;
                        count <=0;
                        nbits<=0;
                    end if;
        end case;
else
        case state is
                when s0 =>
                    if rx = '0' then
                        state <= s1;
                    else
                        state <= s0;
                    end if;
                when s1 =>
                    if count /= 216 then
                        state <= s1;
                        count <= count + 1;
                    else
                        state <= s2;
                        count <= 0 ;
                    end if;
```

```
                when s2 =>
                        if count /= 433 then
                                state <= s2;
                                count <= count + 1;
                        else
                                state <= s3;
                                count <= 0 ;
                        end if;
                when s3 =>
                        if nbits < 8 then
                                state <= s2;
                                buff(6 downto 0)<=buff(7 downto 1);
                                buff(7)<=rx;
                                nbits<= nbits+1;
                        else
                                state <= s5;
                        end if;
                when s5 =>
                        if count /= 433 then
                                state <= s5;
                                count <= count + 1;
                        else
                                state <= s4;
                                count <= 0 ;
                        end if;
                when s4 =>
                        if rx = '1' then
                                count <=0;
                                nbits<=0;
                                state <= s0;
                        end if;
                end case;
        end if;
    end if;
end process;
end rtl;
```

2.6.2. TX

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TX1 is
port(clk_tx,reset_tx,baud_tx,button : in STD_LOGIC;
    tx : in STD_LOGIC_VECTOR (7 downto 0);
            output_tx : out STD_LOGIC
 );
end TX1;


architecture rtl of TX1 is

type state_type is (s5,s6,s7,s8,sA,sB);

signal state : state_type := s5;
SIGNAL count: INTEGER RANGE 0 TO 5199 := 0;
SIGNAL nbits: INTEGER RANGE 0 TO 8 := 0;
SIGNAL buff : STD_LOGIC_VECTOR (7 downto 0);--:="00000000";
signal state_button : std_LOGIC:='1';
signal borda_sub : std_LOGIC := '0';
begin


process(clk_tx,reset_tx)
begin
        if reset_tx ='1' then
                state <=s5;
                buff<="00000000";
                count <=0;
                nbits<=0;
        elsif (rising_edge(clk_tx)) then
                        if baud_tx ='0' then
                                case state is
                                        when s5 =>
```

```
                if borda_sub = '0' then
                        state <= s5;
                else
                        state <= s6;
                        buff<=tx;
                end if;
        when s6 =>
                output_tx <='0';
                state <= s7;
        when s7 =>
                if count /= 5199 then
                        state <= s7;
                        count <= count + 1;
                else
                        state <= s8;
                        count <= 0 ;
                end if;
        when s8 =>
                output_tx <= buff(0);
                buff(6 downto 0)<= buff(7 downto 1);
                nbits<= nbits+1;
                if nbits < 8 then
                state <= s7;
                else
                state <= sA;
                end if;
        when sA =>
                if count /= 5199 then
                        state <= sA;
                        count <= count + 1;
                else
                        state <= sB;
                        count <= 0 ;
                end if;
        when sB =>
                output_tx <= '1';
                buff<="00000000";
                count <=0;
```

```
                        nbits<=0;
                        state <= s5;
                end case;
        else
                case state is
                        when s5 =>
                        if borda_sub = '0' then
                                state <= s5;
                        else
                                buff<=tx;
                                state <= s6;
                        end if;
                when s6 =>
                        output_tx <='0';
                        state <= s7;
                when s7 =>
                        if count /= 433 then
                                state <= s7;
                                count <= count + 1;
                        else
                                state <= s8;
                                count <= 0 ;
                        end if;
                when s8 =>
                        output_tx <= buff(0);
                        buff(6 downto 0)<= buff(7 downto 1);
                        nbits<= nbits+1;
                        if nbits < 8 then
                                state <= s7;
                        elsif nbits = 8 then
                                state <= sA;
                        end if;
                when sA =>
                        if count /= 433 then
                                state <= sA;
                                count <= count + 1;
                        else
                                state <= sB;
```

```
                                        count <= 0 ;
                                end if;
                        when sB =>
                                output_tx <= '1';
                                buff<="00000000";
                                count <=0;
                                nbits<=0;
                                state <= s5;
                        end case;
                end if;
        end if;
end process;

process (clk_tx)
        begin
                if (rising_edge(clk_tx)) then
                        state_button <= button;
                end if;
end process;

borda_sub <= button and not state_button;


end rtl;
```

3. Referências

ROBOCORE TECNOLOGIA. **Comparação entre Protocolos de Comunicação Serial**. [*S. l.*], 2016. Disponível em: https://www.robocore.net/tutorials/comparacao-entre-protocolos-de-comunicacao-serial. Acesso em: 19 out. 2019.

UART - Explicando baudrate e bit de Paridade. [*S. l.*]:YouTube, 2017. Disponível em: https://www.youtube.com/watch?v=xLSgmDTRmCI&t=542s. Acesso em: 19 out. 2019.

#56 - Como funciona a comunicação UART. [*S. l.*]:YouTube, 2018. Disponível em: https://www.youtube.com/watch?v=3PfZ65lnCHw&t=620s. Acesso em: 19 out. 2019.

REZENDE, Roniere. **Interface Gráfica em C# para Comunicação Serial (UART)**. [*S. l.*], 14 jan. 2014. Disponível em: https://www.embarcados.com.br/interface-grafica-para-uart/. Acesso em: 19 out. 2019.

UNIVERSAL asynchroous receiver-transmitter. [*S. l.*], 25 set. 2019. Disponível em: https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter. Acesso em: 19 out. 2019.

BALDASSIN, Alexandro. **Microprocessadores II Comunicação Serial (UART)**. [*S. l.*]. Disponível em: http://www.rc.unesp.br/igce/demac/alex/disciplinas/MicroII/EMA864315-Serial.pdf. Acesso em: 19 out. 2019.

DURDA, Frank. **Tutorial sobre Comunicações Seriais e UART**. [*S. l.*], 18 set. 2018. Disponível em: https://www.freebsd.org/doc/pt_BR/articles/serial-uart/article.html. Acesso em: 19 out. 2019.